DOCUMENTATIE

**TEMA 3**

NUME STUDENT: CÎNDEA ALEXANDRU

GRUPA: 30228

**CUPRINS**

1. Obiectivul temei………………………………………………………….….3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare………………….3
3. Proiectare…………………………………………………………………….3
4. Implementare………………………………………………………………...4
5. Concluzii……………………………………………………………………..14
6. Bibliografie…………………………………………………………………..14

1.Obiectivul temei:

Crearea unei interfete prin care sa putem adauga, modifica, sterge sau vizualiza elementele unor tabele prestabilite si anume client, produs si comanda.

Obiectivele secundare sunt:

1. Analizarea problemei pentru a cunoaste cazurile si scenariile care pot aparea.
2. Proiectarea de diagrame UML pentru a putea structura datele date in clase si interfete.
3. Implementarea reprezinta scrierea codului efectiv si gandirea acestuia intr-un mod cat mai eficient cu putinta.

2.Analiza problemei, modelare, scenarii, cazuri de utilizare:

Se cere implementarea operatiilor de adunare, modificare, stergere si vizualizare pentru elementele tabelelor noastra. In primul rand trebuie creata o interfata care sa selecteze asupra carui tabel dorim sa lucram. De acolo suntem redirectionati catre o interfata care ne permite sa alege operatia care dorim sa o efectuam pe tabelul selectat. Fiecare operatie deschide o noua fereastra independenta din care trebuie citite date sau in care trebuie afisate date. Toate aceste operatii le-am realizat cu ajutorul metodei de reflection care permite scrierea unei singure operatii cu un tip generic care mai apaoi sa poata fi folosita de oricare dintre tabelele pe care dorim sa lucram.

3.Proiectare:

Graphical user interface

Description automatically generated

4.Implenemtarea proiectului:

Acest proiect inglobeaza o paleta larga de clase dar care in mare parte fac acelasi lucru doar ca pentru diferite obiecte. In continuare voi prezenta fiecare pachet:

* Pachetul “Model” poate fi considerat pachetul principal, aici se afla clasele Client, Product si Order care reprezinta obiectele pe care trebuie efectuate operatii.
* Pachetul “Start” este un pachet care contine clasa Main prin care se face inceperea prrogamului si in care se declara primele obiecte de tip interfata pentru si controlerul interfetei principale.
* Pachetul “Presentation” este un pachet care inglobeaza multe alte pacchete foarte importante acestui proiect. In acest pachet se afla pachetele de controller si view. Pachetul “View” contine la randul sau cate un pachet separat pentru fiecare tip de obiect, deci in consecinta acesta contine pachetul “Product”, pachetul “Client” si pachetul “Order”.Fiecare dintre aceste pachete contin in cate un view pentru fiecare operatiune mentionata in cerinta plus un view destinat crearii interfetei pentru selectarea operatiei dorite. Pachetul “Controller” precum pachetul “View” contine cate un pachet pentru fiecare tip de obiect, adica un pachet “Client”, un pachet “Product” si un pachet “Order”. Fiecare dintre aceste pachete contine actinile care trebuie sa se petreaca in momentul in care un buton va fi apasat. Aceste clase sunt destul de similare deci voi prezenta doar clasele din pachetul client:

1. Aceasta este clasa pentru interfata principala

public ClientController(ClientView view,MainView mainView){  
 this.view=view;  
 this.mainView=mainView;  
 this.view.addClientListener(new AddClientListener());  
 this.view.EditClientListener(new EditClientListener());  
 this.view.DeleteClientListener(new DeleteClientListener());  
 this.view.ViewAllListener(new ViewAllListener());  
 this.view.BackMeniuListener(new BackManiuListener());  
}  
  
class AddClientListener implements ActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 AddClientView clientView =new AddClientView();  
 AddClientController addClientController=new AddClientController(clientView);  
  
 }  
}  
  
class EditClientListener implements ActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 EditClientView editClientView=new EditClientView();  
 EditClientController editClientController=new EditClientController(editClientView);  
 }  
}  
  
class DeleteClientListener implements ActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 DeleteClientView deleteClientView=new DeleteClientView();  
 DeleteClientController deleteClientController=new DeleteClientController(deleteClientView);  
 }  
}  
  
class ViewAllListener implements ActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 ViewAllView viewAllView=new ViewAllView();  
 try {  
 ViewAllController viewAllController= new ViewAllController(viewAllView);  
 List<Client> clients= ClientBll.viewAllClient();  
 JTable table=ClientBll.creareTabel(clients);  
 viewAllView.setContentPane(table);  
 } catch (IllegalAccessException ex) {  
 ex.printStackTrace();  
 }  
 }  
}  
  
class BackManiuListener implements ActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 view.dispose();  
 mainView.setVisible(true);  
 }  
}

1. Aceasta este clasa pentru interfata addClient:

public AddClientController*(*AddClientView view*){* this.view=view;  
 this.view.addClientListener*(*new AddClientListener*())*;  
 this.view.BackListener*(*new BackListener*())*;  
  
*}*class AddClientListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* String name=view.getNameField*()*;  
 String email=view.getEmailField*()*;  
 String number=view.getNumberField*()*;  
  
 Client client=new Client*(*name,email,number*)*;  
 ClientBll.*addClient(*client*)*;  
 System.*out*.println*(*"Am scris "+ name+" "+email+" "+number*)*;  
 *}  
}*class BackListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*){* view.dispose*()*;  
 *}  
}*

1. Aceasta este clasa pentru interfata deleteClient:

public DeleteClientController(DeleteClientView view){  
 this.view=view;  
 this.view.DeleteListener(new DeleteListener());  
 this.view.BackListener(new BackListener());  
}  
  
class DeleteListener implements ActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e){  
 int id= Integer.parseInt(view.getIdField());  
  
 ClientBll.deleteClient(id);  
 }  
}  
class BackListener implements ActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e){  
 view.dispose();  
 }  
}

1. Aceasta este clasa pentru interfata editClient:

public EditClientController(EditClientView view){

this.view=view;

this.view.EditareListener(new EditareListener());

}

class EditareListener implements ActionListener{

@Override

public void actionPerformed(ActionEvent e){

int id=Integer.parseInt(view.getIdField());

String name=view.getNameField();

String email=view.getEmailField();

String phone=view.getPhoneField();

Client client=new Client(id,name,email,phone);

ClientBll.updateClient(client,id);

}

}

5.Aceasta este clasa pentru interfata viewAllClients:

public ViewAllController*(*ViewAllView view*)* throws IllegalAccessException *{* this.view=view;  
 this.view.BackListener*(*new BackListener*())*;  
  
*}*class BackListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*){* view.dispose*()*;  
 *}  
}*

Pachetul “Order” contine fata de celelalte pachete doar o clasa deoarece acest obiect nu

Implementeaza toate metodele mentionate anterior. Acest obiect implementeaza doar

metoda de adaugare a unei noi comenzi.

public OrderController*(*OrderView view, MainView mainView*){* this.view=view;  
 this.mainView=mainView;  
 this.view.addOrderListener*(*new OrderListener*())*;  
 this.view.BackMeniuListener*(*new BackMeniuListener*())*;  
*}*class OrderListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*){* int idClient=Integer.*parseInt(*view.getIdClientField*())*;  
 int idProduct=Integer.*parseInt(*view.getIdField*())*;  
 int quantity=Integer.*parseInt(*view.getQuantityField*())*;  
  
 Order order=new Order*(*idClient,idProduct,quantity,0*)*;  
 OrderBll.*addOrder(*order*)*;  
 System.*out*.println*(*"Am scris "+ idClient+" "+idProduct+" "+quantity*)*;  
 *}  
}*class BackMeniuListener implements ActionListener*{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* view.dispose*()*;  
 mainView.setVisible*(*true*)*;  
 *}  
}*

* Pachetul “Connection” aici se afla doar clasa care se ocupa cu crearea legaturii intre codul nostru si baza de date din mySQLWorkbanck.
* Pachetul “Dao”, acesta este cel mai important pachet al programului. In acest program se afla calasa AbstractDao care se ocupa cu implementarea tuturor functiilor pe tipul generic si care totodata recunoaste tipul obiectului pe care noi il transmitem pentru a putea implementa functia pe respectivul obiect. In aceasta clasa avem functiile de insert, update, delete si findAll. Functia de insert pregateste in prealabil un statement care mai apoi sa fie transmis bazei de date. Acesata functie introduce in baza de date un nou obiect. Functia update precateste un statement prin care cauta in baza de date in fuctie de id un anumit element iar apoi ii schimba campurile actuale cu cele noi. Functia delete functioneaza asemanator cu cea de update in sensul ca statementul se executa tot pe baza id-ului dat de utilizator care cand este gasit este sters din bazaa de date. Functia findAll creaza un statement care transmite sub forma unei liste de obicete toate obiectele care se gasesc in tabelul respectiv al bazei de date. Toate aceste functii sunt implementate cu ajutorul claselor din pachetul “Presentation” care a fost discutat anterior si cu ajutorul claselor din pachetul “Bll” care urmeaza sa fie discutat. Tot in acest pachet se afla si clasele ClientDao, ProductDao si OrderDao care doar extind clasa AbstractDao.

public abstract class AbstractDao<T> {  
 protected static final Logger LOGGER = Logger.getLogger(AbstractDao.class.getName());  
  
 private final Class<T> type;  
  
 @SuppressWarnings("unchecked")  
 public AbstractDao(Class<T> type) {  
 // this.type = (Class<T>) ((ParameterizedType) getClass().getGenericSuperclass()).getActualTypeArguments()[0];  
 this.type= type;  
 }  
  
 private String createSelectQuery(String field) {  
 StringBuilder sb = new StringBuilder();  
 sb.append("SELECT ");  
 sb.append(" \* ");  
 sb.append(" FROM ");  
 sb.append(type.getSimpleName());  
 sb.append(" WHERE " + field + " =?");  
 return sb.toString();  
 }  
  
 private String createSelectQueryAll() {  
 StringBuilder sb = new StringBuilder();  
 sb.append("SELECT ");  
 sb.append(" \* ");  
 sb.append(" FROM ");  
 sb.append("`" + type.getSimpleName()+"`");  
 return sb.toString();  
 }  
  
 private String createDeleteQuery(int field) {  
 StringBuilder sb = new StringBuilder();  
 sb.append("DELETE");  
 sb.append(" FROM ");  
 sb.append(type.getSimpleName());  
 sb.append(" WHERE id="+ field);  
 System.out.println(sb);  
 return sb.toString();  
 }  
  
 private String createAddQuery(T object) {  
 StringBuilder sb = new StringBuilder();  
 sb.append("INSERT INTO `");  
 sb.append(type.getSimpleName());  
 sb.append("` VALUES (");  
 try {  
 for (Field field : object.getClass().getDeclaredFields()) {  
 field.setAccessible(true);  
 if (field.get(object) instanceof Integer) {  
 sb.append(field.get(object));  
 sb.append(",");  
 } else {  
 sb.append("'");  
 sb.append(field.get(object));  
 sb.append("',");  
 }  
 }  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 sb.deleteCharAt(sb.length()-1);  
 sb.append(");");  
 System.out.println(sb);  
 return sb.toString();  
 }  
  
 private String createUpdateQuery(T object, int id) {  
 StringBuilder sb = new StringBuilder();  
 sb.append("UPDATE ");  
 sb.append(type.getSimpleName());  
 sb.append(" SET ");  
 try {  
 for (Field field : object.getClass().getDeclaredFields()) {  
 field.setAccessible(true);  
 if (field.get(object) instanceof Integer) {  
 sb.append(field.getName());  
 sb.append("=");  
 sb.append(field.get(object));  
 sb.append(",");  
 } else {  
 sb.append(field.getName());  
 sb.append("=");  
 sb.append("'");  
 sb.append(field.get(object));  
 sb.append("',");  
 }  
 }  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 sb.deleteCharAt(sb.length()-1);  
 sb.append(" WHERE id="+id);  
 System.out.println(sb);  
 return sb.toString();  
 }  
  
 public List<T> findAll() {  
 // TODO:  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createSelectQueryAll();  
 try {  
 connection = ConnectionFactory.getConnection();  
 statement = connection.prepareStatement(query);  
 resultSet = statement.executeQuery();  
  
 return createObjects(resultSet);  
 } catch (SQLException e) {  
 LOGGER.log(Level.WARNING, type.getName() + "DAO:findAll " + e.getMessage());  
 } finally {  
 ConnectionFactory.close(resultSet);  
 ConnectionFactory.close(statement);  
 ConnectionFactory.close(connection);  
 }  
 return null;  
 }  
  
 public T findById(int id) {  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createSelectQuery("id");  
 try {  
 connection = ConnectionFactory.getConnection();  
 statement = connection.prepareStatement(query);  
 statement.setInt(1, id);  
 resultSet = statement.executeQuery();  
  
 System.out.println(resultSet);  
  
 return createObjects(resultSet).get(0);  
 } catch (SQLException e) {  
 LOGGER.log(Level.WARNING, type.getName() + "DAO:findById " + e.getMessage());  
 } finally {  
 ConnectionFactory.close(resultSet);  
 ConnectionFactory.close(statement);  
 ConnectionFactory.close(connection);  
 }  
 return null;  
 }  
  
 public void deleteT(int id) {  
 Connection connection = null;  
 PreparedStatement statement = null;  
 String query = createDeleteQuery(id);  
 try {  
 connection = ConnectionFactory.getConnection();  
 statement = connection.prepareStatement(query);  
 statement.executeUpdate();  
  
  
 } catch (SQLException e) {  
 LOGGER.log(Level.WARNING, type.getName() + "DAO:deletT " + e.getMessage());  
 } finally {  
 ConnectionFactory.close(statement);  
 ConnectionFactory.close(connection);  
 }  
  
 }  
  
 private List<T> createObjects(ResultSet resultSet) {  
 List<T> list = new ArrayList<T>();  
 Constructor[] ctors = type.getDeclaredConstructors();  
 Constructor ctor = null;  
 for (int i = 0; i < ctors.length; i++) {  
 ctor = ctors[i];  
 if (ctor.getGenericParameterTypes().length == 0)  
 break;  
 }  
 try {  
 while (resultSet.next()) {  
 ctor.setAccessible(true);  
 T instance = (T)ctor.newInstance();  
 for (Field field : type.getDeclaredFields()) {  
 String fieldName = field.getName();  
 Object value = resultSet.getObject(fieldName);  
 PropertyDescriptor propertyDescriptor = new PropertyDescriptor(fieldName, type);  
 Method method = propertyDescriptor.getWriteMethod();  
 method.invoke(instance, value);  
 }  
 list.add(instance);  
 }  
 } catch (InstantiationException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 } catch (SecurityException e) {  
 e.printStackTrace();  
 } catch (IllegalArgumentException e) {  
 e.printStackTrace();  
 } catch (InvocationTargetException e) {  
 e.printStackTrace();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (IntrospectionException e) {  
 e.printStackTrace();  
 }  
 return list;  
 }  
  
 public T insert(T t) {  
 // TODO:  
 Connection connection = null;  
 PreparedStatement statement = null;  
 String query = createAddQuery(t);  
 try {  
 connection = ConnectionFactory.getConnection();  
 statement = connection.prepareStatement(query);  
 statement.executeUpdate();  
 } catch(SQLException e) {  
 LOGGER.log(Level.WARNING, type.getName() + "DAO:insert " + e.getMessage());  
 } finally {  
 ConnectionFactory.close(statement);  
 ConnectionFactory.close(connection);  
 }  
 return t;  
 }  
  
 public T update(T t,int id) {  
 // TODO:  
 Connection connection = null;  
 PreparedStatement statement = null;  
 String query = createUpdateQuery(t,id);  
 try {  
 connection = ConnectionFactory.getConnection();  
 statement = connection.prepareStatement(query);  
 statement.executeUpdate();  
 } catch(SQLException e) {  
 LOGGER.log(Level.WARNING, type.getName() + "DAO:update " + e.getMessage());  
 } finally {  
 ConnectionFactory.close(statement);  
 ConnectionFactory.close(connection);  
 }  
 return t;  
 }  
  
 public JTable createTable(List<T> list) {  
 ArrayList<String> columnNames = new ArrayList<String>();  
 int fields = 0;  
 for (Field i : list.get(0).getClass().getDeclaredFields()) {  
 i.setAccessible(true);  
 columnNames.add(i.getName());  
 fields++;  
 }  
 Object[][] data = new Object[list.size()][fields];  
 for (int i = 0; i < list.size(); i++) {  
 int j = 0;  
 for (Field field : list.get(i).getClass().getDeclaredFields()) {  
 try {  
 field.setAccessible(true);  
 data[i][j] = field.get(list.get(i));  
 j++;  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 DefaultTableModel tmodel = new DefaultTableModel(data, columnNames.toArray());  
 JTable table = new JTable(tmodel);  
 table.getTableHeader();  
 System.out.println(table.getTableHeader());  
 return table;  
 }  
  
}

* Pachetul “Bll” este unul din cele mai importante pachete. Acestea contine cate o clasa pentru fiecare obiect . Aceste clase sunt destul de asemanatoare deoarece implementeaza functiile din AbstractDao , adaptandu-le la tipul obiectului care este cerut.

public class ClientBll {  
  
 public static List<Client> viewAllClient(){  
 ClientDao<Client> clientDao=new ClientDao<>(Client.class);  
 return clientDao.findAll();  
 }  
  
 public static void addClient(Client client){  
 ClientDao<Client> clientDao=new ClientDao<>(Client.class);  
 clientDao.insert(client);  
 }  
  
 public static void deleteClient(int id){  
 ClientDao<Client> clientDao=new ClientDao<>(Client.class);  
 clientDao.deleteT(id);  
 }  
  
 public static void updateClient(Client client,int id){  
 ClientDao<Client> clientDao=new ClientDao<>(Client.class);  
 clientDao.update(client,id);  
 }  
  
 public static JTable creareTabel(List<Client> clients){  
 ClientDao<Client> clientDao=new ClientDao<>(Client.class);  
 return clientDao.createTable(clients);  
 }  
  
}

Clasa OrderBll precum am mentionat anterior este putin diferita deoarece nu implementeaza toate metodele implementate in celelalte clase. Aceasta implementeaza doar metoda de adaugare:

public class OrderBll *{* public static void addOrder*(*Order order*){* OrderDao*<*Order*>* orderDao=new OrderDao*<>(*Order.class*)*;  
 orderDao.insert*(*order*)*;  
 *}  
}*

5.Concluzii:

In opinia mea aceasta tema este una muncitoreasca cu multe concepte repetitive. In aceasta tema inveti cum sa folosesti mai bine o clasa abstracte, te invata cum sa creezi o conexiune intre un program de tip java si o baza de date. Este prima data cand folosesc o clasa cu parametrii de tip generici care pot fi inlocuiti ulterior cu un obiect anume , obiect recunoscut automat tot de acelasi program. Implementarea acestei teme a fost destul de dificila si cu multe provocari. Proiectul meu poate fi imbunbatatit prin implementarea formulei pentru pretul total al unei comenzi, totodata scazand stocul produsului in momentul introducerii acestuia in comanda. Mai poate fi imbunatatit si prin adaugarea unei “Facturi” care sa fie pusa intr-un fisier text si salvata pentru fiecare comanda nou plasata.

6.Bibliografie:

<https://dsrl.eu/courses/pt/materials/A3_Support_Presentation.pdf>

https://stackoverflow.com/

d d d d d d d d d d d d d d d d d d d d d d d dd d d a a a a d d d a a a d d d a a d d d a a d d d a a d a d